AN ITERATIVE ALGORITHM FOR DELAY-CONSTRAINED MINIMUM-COST MULTICASTING

Mehrdad Parsa, Qing Zhu and J.J. Garcia-Luna-Aceves
Computer Engineering Department
School of Engineering
University of California
Santa Cruz, CA 95064
courant, qingz, jj@cse.ucsc.edu

ABSTRACT

The bounded shortest multicast algorithm (BSMA) is presented for constructing minimum-cost multicast trees with delay constraints. BSMA can handle asymmetric link characteristics and variable delay bounds on destinations, specified as real values and minimizes the total cost of a multicast routing tree. Instead of the single-pass tree construction approach used in most previous heuristics, the new algorithm is based on a feasible-search optimization strategy that starts with the minimum-delay multicast tree and monotonically decreases the cost by iterative improvement of the delay-bounded multicast tree. BSMA's expected time complexity is analyzed, and simulation results are provided showing that BSMA can achieve near-optimal cost reduction with fast execution.

Keywords: multicast, multimedia, spanning tree, minimum-cost tree.

*This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contracts F19628-93-C-0175 and F19628-96-C-0038. This paper has been presented in part at the IEEE INFOCOM'95 Conference, Boston, MA.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1998		**00-00-1998 to 00-00-1998**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
An Iterative Algorithm for Delay-Constrained Minimum-Cost Multicasting	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**41**	

1 Introduction

Multicasting consists of concurrently sending the same information from a source to a subset of all possible destinations in a computer network. Multicast service is becoming a key requirement of computer networks supporting multimedia applications. To carry large numbers of multicast sessions, a network must minimize the sessions' resource consumption, while meeting their quality of service (QOS) requirements. The current approach for efficiently supporting a multicast session in a network consists of establishing a multicast tree for the session, along which session information is transferred. Algorithms are needed in the network to compute multicast trees; we call such algorithms *multicast algorithms*.

Different optimization goals can be used in multicast algorithms to define what constitutes a good tree. One such goal is providing the minimum delay from source to destinations along the tree, which is important for delay-sensitive multimedia applications, such as real-time teleconferencing. Another optimization goal is constructing the minimum cost tree, which is important in managing network resources efficiently. The tree cost is defined as the total cost of the links of the tree. We call this objective *utilization-driven*, because it minimizes the total utilization of links.

Optimization techniques for multicasting proposed before have considered the delay and cost optimization objectives, but have treated them as distinct problems. Dijkstra's shortest path algorithm [8] can be used to generate the shortest paths from the source to destinations. This provides the optimal solution for delay minimization. Multicast algorithms that perform cost optimization have been based on computing the minimum Steiner tree in a graph. A Steiner tree in a graph must reach a subset of nodes in the graph, called *given* nodes. The problem of finding the minimum Steiner tree is known to be an NP-complete problem [12], and a number of heuristics [15, 19, 22] have been developed to solve this problem in polynomial time and producing near-optimum results. In Kou, Markowsky and Berman's (KMB) algorithm [15], a network is abstracted to a complete distance graph consisting of edges that represent the shortest paths between the source node and each destination node. The KMB algorithm constructs a minimum spanning tree [18] in the complete distance graph, and the Steiner tree of the original network is obtained by achieving the shortest paths represented by edges in the minimum spanning tree.

Bharath-Kumar and Jaffe [3] discussed minimizing both cost and delay, assuming that cost and delay functions are identical. Awerbuch, Baratz and Peleg [1] proposed a heuristic (which we call ABP) that starts from the minimum spanning tree and refines the tree to bound the diameter of the tree. ABP applies to only spanning trees and not Steiner trees. This same idea is used in the work by Cong, Kahng, Robins, Sarrafzadeh and Wong [6] (which we call CKRSW) except that CKRSW refines the tree to bound the radius of the tree. Both ABP and CKRSW assume identical cost and delay functions. Kompella, Pasquale and Polyzos [14] proposed two heuristics (which we call KPP) that address delay-bounded multicast trees. In their formulation, the delay bound for all destinations is the same; furthermore, KPP assumes that link delays and the delay bound are integer-valued and that link costs and delays are symmetric. KPP extends the KMB algorithm by taking into account the constraint of the specified delay bound in the construction of the complete distance graph [14].

This paper presents a new algorithm for the construction of a minimum-cost multicast tree with delay constraints. The algorithm, which we call *Bounded Shortest Multicast Algorithm* (BSMA)[1] proceeds in two phases. It starts by obtaining a minimum delay tree using any of the well known shortest path algorithms. (We use Dijkstra's algorithm in our implementation). Starting from this tree, BSMA iteratively improves the cost of the delay constrained tree. BSMA has several novel features with respect to prior approaches to the delay-constrained minimum-cost multicasting problem:

- Non-uniform, positive and real-valued delay bounds are accepted. When destinations have different delays in receiving messages from the source, variable delay bounds of destinations are necessary. The capability to specify arbitrary delay bounds allows BSMA to model delay-bounded multicast instances more accurately, and to always find a feasible solution. In contrast, KPP assumes a unique integer-valued delay bound for all destinations.

- The delay function on links can take arbitrary positive real values. This ability enables BSMA to model networks more accurately, and to always find a feasible solution. In contrast, KPP assumes an integer-valued delay function.

[1]The fact that BSMA also spells "Banana Slug Multicast Algorithm" and that the authors' affiliation is UCSC is entirely coincidental, of course.

- Instead of using a single pass to construct the tree as in previous algorithms, BSMA employs a multiple-pass approach to iteratively minimize the cost function of the tree. The iterative nature of BSMA allows it to trade-off cost performance versus running time.

- BSMA can test the feasibility of a delay-bounded multicast instance with delay bound Δ in an n-node graph in $O(n^2)$. In contrast, KPP requires $\Omega(\Delta n^3)$.

- BSMA accepts asymmetric link costs and delays. The ability to accept asymmetric link costs and delays makes BSMA more versatile and applicable in actual networks. Because actual networks must be considered as directed graphs when bandwidth and delay must be managed independently in each direction on a link. In contrast, KPP assumes symmetric link costs and delays.

Consistent with the assumptions made in all prior approaches to delay-constrained minimum-cost multicasting, throughout this paper we assume that a node running BSMA has complete topology information. The algorithms and associated protocols needed to deliver such information at each node are beyond the scope of this work. Section 2 describes the multicasting model assumed in BSMA. Section 3 describes BSMA and proves the correctness of BSMA. Section 4 gives an analysis of BSMA's time complexity. Section 5 presents simulation results, indicating that BSMA achieves near-optimum cost reduction and fast execution. Section 6 gives our concluding remarks.

2 Delay-Constrained Multicast Model

The source node of a multicast is assumed to know all the information needed to construct the multicast tree. One way to have this information is for each node in the network to maintain an updated database of link weights and node connections in the network. This requirement can be supported using one of many topology-broadcast algorithms, which can be based on flooding or other techniques [2, 11, 10].

A network is modeled as a directed weighted graph $G = (V, E)$, as shown in Figure 1, with node set V and edge set E. The nodes represent routers or switches and edges represent the communication link between them. An edge $e \in E$ from $v \in V$ to $u \in V$ is

represented by $e = (v, u)$. A path in G is a sequence $\langle v_0, v_1, ..., v_k \rangle$ of vertices such that edge $(v_{i-1}, v_i) \in E$ for $i = 1, 2, ..., k$. The path contains the vertices $v_0, v_1, ..., v_k$ and the edges $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$. A path is simple if all vertices in the path are distinct. The nodes $v_1, ..., v_{k-1}$ in a path $\langle v_0, v_1, ..., v_k \rangle$ are called internal nodes.

Nodes in V can be of the following three types:

- **Source node**: a node connecting to the source that sends out the data stream
- **Destination node**: a node connecting to a destination that receives the data stream
- **Relay node**: any node that is neither a source node nor a destination node

Two positive real-valued functions are defined on E:

- **Link-Cost Function** ($c : E \rightarrow \Re^+$): The cost of a link, which can be associated with the utilization of the link. A higher utilization is represented by a higher link cost.
- **Link-Delay Function** ($d : E \rightarrow \Re^+$): The delay of a link is the sum of the perceived queueing delay, transmission delay, and propagation delay over that link.

Let s be the source node, and D the set of destination nodes. For each path from s to a destination node $v \in D$, the delay of the path, or *path delay*, is defined to be the sum of link delays along the path. A set of path-delay upper bounds are assigned to destinations by the following function:

Destination Delay-Bound Function or DDF ($\delta : D \rightarrow \Re^+$): DDF assigns an upper bound to the path delay from the source to each destination in the multicast tree. Of course, $\delta(i)$ can be different from $\delta(j)$ for destinations $i \neq j$. In the special case that DDF assigns the unique delay bound to all destinations, this delay bound is denoted by $\delta(i) = \Delta, \ \forall i \in D$.

DDF determines the delay constraints that the multicast tree is required to satisfy, while the cost of the multicast tree is minimized. When different delay bounds are assigned on destinations, the topology of the multicast tree has to be updated to meet the delay bounds. The multicast tree that we are interested in constructing is a delay-bounded minimum Steiner tree (DBMST) and the minimization problem can be formally described as follows:

DBMST Problem: *Given a graph $G = (V, E)$ with a link-cost function, a link-delay function, a source s, a set of destinations D, and a DDF, then construct a DBMST spanning*

$D \cup \{s\}$, such that the cost of the tree is minimized while DDF is satisfied. The cost of the multicast tree is defined as the sum of link costs in the multicast tree.

Consistent with prior work on delay-constrained multicasting, how the delay and cost of the links and the destination delay bounds are obtained are outside the scope of this work. The DBMST problem can be reduced to a minimum Steiner tree problem with the delay bound set to infinity, which is known to be an NP-complete problem [12] and only heuristics are of practical interest for its solution; BSMA is a new heuristic designed to solve the DBMST problem.

3 Description of BSMA

The DBMST can be approached as a feasible-search optimization problem [4] in which the feasible region consists of all trees that satisfy the delay bound requirement. BSMA constructs a DBMST in two phases, as illustrated in Figure 2:

1. Initial phase: construct an initial tree with the minimum delays from the source to all destinations.

2. Improvement phase: iteratively minimize the cost of the tree while always satisfying the delay bounds.

To guarantee that a feasible solution is found that satisfies the given delay bound, the initial tree is the minimum-delay tree, which is constructed using Dijkstra's shortest-path algorithm [8]. In some cases, the delay bounds given by DDF may be too tight, i.e., they cannot be met even in the minimum-delay tree. In such cases, some negotiation is required to relax the delay bounds of DDF before any feasible tree can be constructed, as shown in the BSMA flowchart (Figure 2). The bounds given by DDF must be relaxed until they can be met by the minimum-delay tree. The rest of this paper assumes that DDF assigns the delay bounds that can be met by the minimum-delay tree. A high-level specification of BSMA is shown in Figure 4.

BSMA's improvement phase iteratively transforms the tree topology to decrease its cost monotonically, while satisfying the delay bounds. Let T_0 denote the initial tree topology. During the improvement phase, at iteration j the current tree T_j is transformed to tree T_{j+1},

and at iteration $j+1$ the tree T_{j+1} is transformed to tree T_{j+2}, and so on. The sequence of trees $T_0, T_1, \ldots, T_{final}$ is such that $cost(T_0) \geq cost(T_1) \geq \ldots \geq cost(T_{final})$, where $cost(T_j)$ is the objective function being minimized. Delay bounds are satisfied throughout the iterative improvement.

3.1 Delay-Bounded Path-Switching

The transformation performed by BSMA at each iteration of the improvement phase consists of a delay-bounded path switching, by which a path in T_j is replaced by a new path not in T_j with smaller cost resulting in a new tree topology T_{j+1} [25]. Ensuring an effective delay-bounded path switching improvement involves the following:

1. Choosing the path to be taken out of T_j, obtaining two disjoint subtrees ${T_j}^1$ and ${T_j}^2$.
2. Finding a new path to connect ${T_j}^1$ and ${T_j}^2$, resulting in the new tree topology T_{j+1} with smaller cost, while the delay bounds are satisfied.

A candidate path in T_j for path-switching improvement is called a superedge.

Definition (Superedge): *A superedge is a simple path $\langle v_0, v_1, ..., v_k \rangle$ such that all internal nodes, i.e., $v_1, ..., v_{k-1}$, are relay nodes which connect exactly two tree edges, and is not contained in a longer path with the same property.*

A superedge consists of one or more tree edges and zero or more internal tree nodes. Removing a superedge from a multicast tree corresponds to removing all the tree edges and internal nodes in the superedge. From the definition of a superedge, only relay nodes can be internal. A destination node or a source node cannot be an internal node of a superedge. This prevents the removal of destination nodes or the source node from the tree as a result of removing a superedge.

Definition (Cost of superedge): *The cost of superedge is defined as the* sum *of link costs along the simple path that corresponds to the superedge.*

As shown in Figure 4, all superedges are initialized as unmarked. The superedge p_h with the highest cost among all *unmarked* superedges is selected to be removed. This superedge selection for path-switching is called the naive heuristic. Removing the highest-cost superedge p_h in T_j breaks T_j into two disjoint subtrees ${T_j}^1$ and ${T_j}^2$, where $T_j = p_h \cup {T_j}^1 \cup {T_j}^2$. A delay-bounded shortest path p_s between T_j^1 and T_j^2 is used to connect T_j^1 and T_j^2 to obtain

the new tree topology T_{j+1}, i.e., $T_{j+1} = p_s \cup T_j^1 \cup T_j^2$. A delay-bounded shortest path is defined as the path with the smallest cost, subject to the constraint that $T_{j+1} = p_s \cup T_j^1 \cup T_j^2$ is a delay-bounded tree. Clearly, the cost of p_s is not higher than the cost of p_h.

The search for the delay-bounded shortest p_s path used to reconnect the two trees T_j^1 and T_j^2 starts with the shortest path between the two trees. If, however, the shortest path results in a violation of delay bounds, BSMA uses an incremental k-shortest path algorithm [16, 24] to find a delay-bounded shortest path to reconnect the two trees. The k-shortest path problem consists of finding the kth shortest simple path connecting a given source-destination pair in a graph. Note that, in BSMA, the k-shortest path is between two *trees*. To find the k-shortest path between two trees, a standard technique introduces two additional nodes: one connected to all the nodes in one tree and the other connected to all the nodes in the other trees, with all the new connections having zero cost. Then, finding the k-shortest path between the two trees is equivalent to finding the k-shortest path between the two new nodes.

The value of k is known a posteriori, i.e., after the delay-bounded shortest path is found. The incremental construction of the k-shortest paths between T_j^1 and T_j^2 proceeds with the construction of the 1st, 2nd, ..., kth shortest-path, where k is the smallest value for which either of the following is true:

- The k-shortest path has a smaller cost than the removed superedge and satisfies the delay bounds
- The k-shortest path has the same cost as the removed superedge

It is possible for k to be very large for some instances of networks. However, the value that k can take can be controlled by making k an input to BSMA. This way, BSMA can trade off cost performance versus running time. This matter is further discussed in Section 4.

One of two cases must happen when p_s is obtained:

(a) Path p_s is the same as the path of p_h; or

(b) Path p_s is different from the path of p_h.

Finding the delay-bounded shortest path always terminates, because the path of the deleted superedge p_h is found again in the worst case. If case (a) occurs, BSMA marks p_h to indicate

that p_h has been examined without improvement of the tree cost. BSMA continues the path-switching by examining the next unmarked superedge in T_j with the highest cost. If case (b) occurs, BSMA unmarks all marked superedges in T_j, and continues to do the path switching for the highest-cost unmarked superedge.

In Figure 3(a), the superedges in the order of decreasing path cost are $\langle s, d, c\rangle$, $\langle s, g\rangle$, $\langle s, f\rangle$, and $\langle f, e\rangle$. The highest-cost superedge $\langle s, d, c\rangle$ is deleted from the tree, resulting in two disjoint trees. One tree consists of nodes e, f, g and s and the other tree consists of the singleton c. A delay-bounded shortest path $\langle f, a, c\rangle$ is used to reconnect the two trees to get tree T_1, as shown in Figure 3(c).

BSMA terminates when all superedges become marked, which occurs when all possible superedges in the tree have been examined for path-switching without success in reducing the tree cost. The significant cost minimization of multicast tree by BSMA is demonstrated by the simulation results in Section 5. The next theorem shows the correctness of BSMA in finding a delay-bounded multicast tree.

Theorem 1: *BSMA always finds a delay-bounded Steiner tree if it exists.*

Proof: The proof is by induction on j which is the iteration number of tree topologies. T_0 is a minimum-delay tree. If the minimum-delay tree T_0 cannot satisfy the specified delay bounds DDF, a delay-bounded tree cannot exist for the specified DDF. When T_j is transformed to T_{j+1} by the delay-bounded path-switching, the delay-bounded shortest path between the two trees T_j^1 and T_j^2 by definition implies that the new tree T_{j+1} satisfies the delay bounds. Finding the delay-bounded shortest path always terminates, because in the worst case the path of the deleted superedge in T_j is found again. □

3.2 Example of Operation

This section gives a small example showing the initial and improvement phases of BSMA. The network for the example is given in Figure 3. Notice that each link in the network has a cost and a delay in each direction. The source is represented by solid black disk and the destination nodes are solid black squares.

Figure 5 illustrates the steps of BSMA for utilization-driven tree construction and minimization. The source node has complete topology information. Each link is represented by two directed edges, one for each direction, and each direction has a cost and delay associated

with it. Each destination has a delay bound for the multicast indicated by $\{delay\_bound\}$. Notice that delay bounds are different real-values for different destinations. Although the cost of a link can be asymmetric, we have set the same link costs in both link directions to simplify the examples. The link costs and delays are shown $cost, delay$ pairs in the figures.

Initially, the minimum delay tree T_0 is constructed as shown in Figure 5(a). This initial tree is iteratively improved by selecting a superedge and performing path-switching. The overall cost reduction is 40% in the improvement phase. During the improvement phase, the trees T_1, T_2, and T_3 are constructed after the first, second and third iteration, respectively, such that they satisfy the delay bounds, as shown in Figures 5(b)-(d), The final tree is shown in Figure 5(d).

3.3 A Greedy Variant

A more sophisticated heuristic to use in superedge selection for path-switching is based on a greedy choice for cost reduction. Let p be a superedge in tree T_j, and q be the corresponding delay-bounded shortest path used to reconnect the tree when p is removed from the tree. Denote the path cost of p and q as c_p and c_q, respectively. The *gain* g of path-switching p to q is defined to be $g = c_p - c_q$. Then, the greedy choice is to select the superedge in the current tree T_j which gives the maximum gain g^* for path-switching. BSMA terminates when the maximum gain g^* is zero. A high-level description of BSMA based on the greedy heuristic is given in Figure 6.

In the greedy heuristic, path-switching is performed on each superedge p tentatively to find the corresponding delay-bounded shortest path q, and the gain. This is done independently for each superedge; thus, the current tree stays the same for each tentative path-switching. After evaluating all superedges in the current tree T_j, The superedge with the maximum gain is selected for definite path-switching to transform the current tree.

Empirical results are obtained in Section 5 to compare the tree costs of the greedy heuristic with the naive heuristic given in Section 3.1. We found the difference in costs for final trees to be quite small, i.e., less than 1% for all tested examples. Although the greedy heuristic may make the behavior of BSMA clearer or more obvious, it has a higher time complexity, without significantly lower cost than the naive heuristic. By evaluating all superedges in the current tree for path-switching, the time complexity of the greedy heuristic

per tree transformation is higher by a factor $O(|U|)$, where U is the set of superedges and $|U| = O(n)$ for a network with n nodes.

3.4 Adapting to Changes

The heuristic described in this paper can be incorporated into an appropriate multicast protocol. This, however, would require BSMA to adapt to changes to the multicast tree. To make BSMA adapt to changes in the network topology or group membership, BSMA can be easily extended to build delay-constrained multicast trees dynamically. The steps of BSMA for the dynamic construction of a delay-bounded multicast tree are as follows:

- To add a destination x to the multicast tree.
  - Step 1: Connect x to the source by the minimum delay path from the source.
  - Step 2: Iteratively perform path-switching to lower the cost of the tree.
- To remove a destination x from the multicast tree.
  - Step 1: If x is connected to more than two edges in the multicast tree, make the node as relay node. Else, delete the destination node along with its connecting superedge from the tree.
  - Step 2: Iteratively perform path-switching to lower the cost of the tree.

When adding a new destination node, the delay bound is first satisfied in Step 1 as in BSMA. Step 2 then iteratively refines the new tree for low cost using the delay-bounded path switching improvement technique described in this paper. Note that the multicast tree is *not* re-computed each time a destination is added or removed. The fact that the multicast tree is not completely rebuilt after each membership change makes BSMA a practical candidate for solving the on-line variation of the DBMST Problem. When multiple destinations are added and/or removed simultaneously, Step 1 can be performed for these destinations, then the tree is iteratively refined in Step 2 to reduce the tree cost.

4 Time Complexity Analysis

The complexity of BSMA is $O(pt)$, where p is the number of path-switching operations and t is the complexity of a path-switching operation. In this section, we examine the complexities

of p and t.

The value of p can be an input to BSMA. It can be a fixed number or a function of the number of nodes or superedges as desired. We use a stochastic model to get an expression for the *expected* number of path-switching operations performed by BSMA, when p is *not* specified as an input. In order to derive the expectation, we make a probabilistic assumption about the tree transformation process: A given tree is equally likely to be transformed to any monotonically less costly tree. We show that the expected time complexity of BSMA when using naive heuristic is $O(kn^3 \log(n))$ for general graphs, and $O(kn^3)$ for degree-bounded graphs. The complexity is increased by a factor of n when using the greedy heuristic of Section 3.3.

Lemma 1: *The expected number of path-switchings performed by BSMA when using the naive path-switching heuristic is* $O(n \log(n))$ *for a general network with* n *nodes.*

Proof: To analyze the maximum number of path-switchings done, we examine the behavior of an idealized algorithm, called IA, which can find the optimal solution of the DBMST problem. IA is used to model the iterative behavior of BSMA. IA constructs an initial Steiner tree and iteratively transforms the tree such that the tree cost is monotonically decreased to reach the global minimum. Note that we are not concerned with how IA transforms the tree. The expected number of path-switchings is found by constructing a Markov chain, where each state corresponds to a Steiner tree. The state transition corresponds to possible transformation of one Steiner tree into another. In order to account for unsuccessful path-switchings, each state is replicated $n-1$ times, as there are at most $n-1$ superedges. Each of these n replicated states corresponds to the same tree. An unsuccessful path-switching, which is called an identity transformation, corresponds to a state transition from a state into one of its replicated states. By construction, IA may perform an identity transformation on a tree for a maximum of $n-1$ iterations.

By Cayley's Theorem [5], there are n^{n-2} possible spanning trees on n nodes. The number of distinct Steiner trees is no greater than the number of spanning trees. The number of Steiner trees is then bounded by n^{n-2}. Construct a Markov chain of n^{n-2} states, where each state corresponds to a spanning tree. Sort these states with respect to the cost of the Steiner tree (not the spanning tree) breaking ties arbitrarily. Replace each state with n copies of itself to get a total of n^{n-1} states. Number the states sequentially, starting from 1 at the

cheapest cost state to n^{n-1} at the most expensive state.

In this Markov chain, transition edges go from a state S_i to a state S_j, such that $j < i$. It is assumed that each of the possible transitions from a state is equally likely in IA. Thus, the probability of transition from S_i to S_j is

$$P_{ij} = \frac{1}{i-1} \text{ for } 1 \leq j < i$$

and $P_{11} = 1$. Let T_i be the number of transitions needed to go from state i to state 1. The expected value can be found by conditioning on the first transition from a given state. Let Y be the random variable of the next state of the first transition.

$$\begin{aligned} E[T_i] &= E[E[T_i|Y]] = \sum_y E[T_i|Y=y]P\{Y=y\} \\ &= \sum_y E[T_i|Y=y]\frac{1}{i-1} \\ &= \frac{1}{i-1}\sum_{y=1}^{i-1}(1+E[T_y]) \\ &= 1 + \frac{1}{i-1}\sum_{y=1}^{i-1}E[T_y] \end{aligned}$$

Using induction with $E[T_1] = 0$, it can be shown that $E[T_i] = \sum_{y=1}^{i-1} 1/y \approx \log(i)$.

Therefore, if IA starts in the most expensive state, i.e., $i = n^{n-1}$, then the expected number of transitions, i.e., path-switchings, is $O(\log(n^{n-1})) = O(n\log(n))$. This is for IA, which by assumption can always find the global minimum. BSMA is likely to terminate earlier at a local minimum, i.e., when path-switching fails consecutively for every superedge. Thus, the maximum expected number of path-switchings done by BSMA is also $O(n\log(n))$. □

We will next examine the complexity of each path-switching operation. In our description and implementation of BSMA, we have used a k-shortest-path algorithm to perform path-switching. The complexity of k-shortest path algorithm[2] is $O(ks(n))$, where $s(n)$ is the complexity of single-source shortest-path algorithm [13]. However, k can be exceedingly large. For example, consider a grid graph, where all edges have a unit cost, and some delay. The number of paths between two nodes grows exponentially with the Manhattan distance

[2]Our implementation of the k-shortest path uses the algorithm described by Lawler [16] which takes $O(kn^3)$.

between the nodes, all of which have the same cost. Since a k-shortest path algorithm between two subtrees enumerates all paths of the same cost, if the shortest path connecting the two subtrees does not satisfy the delay bounds, then k can become very large.[3] When using the k-shortest-path path-switching, the running time of BSMA can be traded against the cost-performance by giving an upper bound on k as an input of BSMA (see Section 5). Letting k be a small value will be faster but the cost may not be as low, and vice versa. By fixing k, the running time of path-switching becomes a polynomial in n only. However, in our experiments, we have observed the algorithm to be pretty fast (see Section 5 for average execution times).

Theorem 2 gives the time complexity of BSMA when using the naive superedge selection heuristic and k-shortest-path path-switching. In the following theorems and corollaries, we let k be the average number of shortest paths evaluated to get the delay-bounded shortest path.

Theorem 2: *The expected time complexity of BSMA when using the naive superedge selection is $O(kn^3 \log(n))$ for a general network with n nodes.*

Proof: The computational time in every round of path-switching is dominated by the k-shortest path algorithm, which uses a single-source shortest-path algorithm. A straightforward implementation of Dijkstra's algorithm has $O(n^2)$ complexity. Thus, $s(n) = O(n^2)$. Using more sophisticated data structures $s(n)$ can be reduced to $O(m + n \log n)$, where m is the number of edges in the network [9]. This complexity is better for sparse graphs. By Lemma 1, the expected number of path-switchings to be performed is $O(n \log(n))$. Thus, the total time complexity of BSMA is $O(ks(n) \cdot n \log(n)) = O(kn^2 \cdot n \log(n)) = O(kn^3 \log(n))$. □

In practice, we are more interested in degree-bounded networks in which the maximum degree of every node is upper-bounded and typically much smaller than the total number of nodes.

Lemma 2: *The expected number of iterations performed by BSMA when using the naive superedge selection is $O(n)$ for a degree-bounded network with n nodes.*

[3] One way to limit k is to examine the shortest-paths from the nodes of one subtree (T_j^1) to the nodes of the other (T_j^2). In the worst case, there are $n/2$ nodes in each subtree, thus, there are $O(n^2)$ shortest-paths to consider. The complexity of this method for path-switching is $O(n^3)$, which is dominated by performing Dijkstra's algorithm n times. We will not consider this method for path-switching any further, and will focus on the k-shortest-path algorithm for path-switching.

Proof: The number of edges is upper-bounded by $dn/2$ for a degree-bounded graph with the maximum degree d and n nodes. Therefore, by basic counting, the maximum number of possible graphs is $2^{dn/2}$, which gives an upper bound on the number of spanning trees. Following the same line of proof in Lemma 1, we get the expected number of path-switchings to be $O(\log(n2^{dn/2})) = O(n)$. □

Lemma 2 leads to the time complexity of BSMA for a degree-bounded network.

Theorem 3: *The expected time complexity of BSMA when using the naive superedge selection is $O(kn^3)$ for a degree-bounded network with n nodes.* □

For the greedy superedge selection, every superedge in the tree is evaluated to find the superedge that gives the maximal gain for cost reduction. The number of superedges in the tree is $O(n)$. For a general network, using a similar construction as in Lemma 1, but without replicated states, we can see that the number of path-switchings using the greedy heuristic is also $O(n\log(n))$. However, each path-switching involves evaluating k-shortest-path for $O(n)$ superedges, taking $O(kn^2 \cdot n) = O(kn^3)$ time. Thus, we have the following corollaries.

Corollary 1: *The expected time complexity of BSMA when using the greedy superedge selection is $O(kn^4\log(n))$ for a general network with n nodes.* □

Corollary 2: *The expected time complexity of BSMA when using the greedy superedge selection is $O(kn^4)$ for a degree-bounded network with n nodes.* □

5 Simulation Results

BSMA has been implemented in C++. The experiments were carried out using the Arpanet topology and sparsely connected random graphs. We used the Arpanet topology to illustrate BSMA's behavior in a topology of what once was a real network. The random graphs used were designed to be sparse with the average degree being less than five, simply to capture the flavor of network topologies in which links between nodes are still relatively expensive commodities. As we expected, we obtained qualitatively similar results for all the different graphs. Group members were picked uniformly from the set of nodes in the graph, excluding the nodes already selected for the group.

The random graphs used in the simulation are constructed using the method proposed by Waxman [23]. The n nodes of a graph are randomly placed on a Cartesian coordinate grid with unit spacing. The (x, y) coordinates of each node was selected uniformly from

integers in $[0, n]$. Considering all possible pairs of node, edges are placed connecting nodes with probability

$$P(u, v) = \beta \exp\left(\frac{-d(u, v)}{\alpha L}\right)$$

where $d(u, v)$ is the Manhattan distance between nodes u and v, and L is the maximum possible distance between two nodes. The parameters α and β are in the range (0,1] and can be selected to obtain desired characteristics in the graph. For example, a large β gives nodes with a high average degree, and a small α gives long connections. It has been observed that, with appropriate parameters, this method gives networks that resemble "real world" networks. The parameters α and β are varied to obtain appropriately sparse networks, i.e., average degree of node is less than or close to five.

The cost of each edge was set to the Manhattan distance between its endpoints plus one. By adding one to the Manhattan distance, the case of zero edge cost is eliminated. The delay of an edge is set to a uniform random number in [0,1] times its cost plus one. This definition of delay is used to eliminate the unrealistic possibility of zero delay. The graphs obtained for the simulation runs have the average degree listed in Table 1.

BSMA was run on 25-, 50-, 75-, and 100-node graphs. Three different sizes were used for the number of destinations in the runs: 4, 6, and $|V| - 1$. For the sake of simplicity, the delay bound is the same for all destinations (but it is not necessarily an integer). The cost of each resulting multicast tree is normalized by the cost of the KMB algorithm for the same group instance. The resulting cost ratio ρ is averaged over the number of groups $|M|$, i.e.,

$$\rho = \frac{1}{|M|} \sum_{i=1}^{|M|} \frac{cost(T_{BSMA})}{cost(T_{KMB})}$$

The total number of groups is 2500 and 3500 for the set of runs with 4 destinations and 6 destinations, respectively. The total number of groups with $|V| - 1$ destinations is $|V|$.

The results using naive superedge selection are shown in Figure 7. The error bars represent 95% confidence intervals. The cost of each sample is normalized by the cost of the *unconstrained* solution obtained using the KMB algorithm. The label `bsm-d` is assigned to the cost ratio of the minimum delay solution obtained using Dijkstra's algorithm. This is the starting solution of BSMA. For the sake of simplicity and comparison, the delay bound is the same for all destinations. The label `bsm-1` is given to the cost ratio of BSMA with

the delay bound equal to the maximum delay in the KMB solution. The label `bsm-0` corresponds to the cost ratio of BSMA with the delay bound equal to the maximum delay in the minimum delay solution. The label `bsm-1/2` is assigned to the cost ratio of BSMA with the delay bound half way between the two extremes of the KMB and minimum-delay solutions. The cost ratio of the BSMA solution when the delay bound is the same as the maximum delay of the KMB solution indicates that the cost of BSMA solution is actually *less* than the unconstrained KMB solution.[4] We suspect that the superiority of BSMA over KMB is due to the iterative nature of BSMA. The KMB solution has been shown to yield a worst-case cost performance of $2(1 - \frac{1}{|Z|})$, where Z the set of leaves in the optimal Steiner tree [15]. In practice, the average suboptimality of KMB is far below this limit and is near optimal. Therefore, we conjecture from our experiments that the quality of a solution found by BSMA is near-optimal in practice.

We expect BSMA to obtain smaller costs than those of KPP, because KPP is based on the KMB algorithm. Recent simulation results obtained by Salama et al. [21] confirm that the cost of BSMA is consistently lower than the cost of KPP; their results also show that BSMA is better than KPP and all the other algorithms reported to date for delay-constrained multicasting. Moreover, our experiments and the simulation results by Salama et al. [21] confirm that BSMA *always* succeeded in constructing a tree if there is a feasible solution, whereas KPP can fail if the granularity used to scale costs to integer values is not of proper size.

Using the tightest possible delay bound, as determined by the minimum-delay tree, the cost ratio indicates that the cost of BSMA tree is substantially better than the cost of the minimum-delay tree. This is because the non-zero delay slack (i.e., difference between delay bound and actual delay) of some destinations can be used to reduce the cost of the tree. By controlling the delay bound between the two extremes, namely, the KMB and minimum-delay solutions, a range of minimum-cost solutions can be obtained.

Another set of runs was done with all the nodes in the network belonging in one group and each node taking turn as the source and the rest of the network being the set of destinations. In this case, the Steiner tree problem reduces to the minimum-weight spanning tree for the

[4] The previous published results in INFOCOM '95 showed the cost of KMB as being less, because path-switching was also applied to the result of KMB as a postprocessing, which further reduced the cost of KMB trees.

unconstrained case. Finding the minimum-weight spanning tree can be solved optimally in $O(n^2)$ using Prim's algorithm [18]. The results are shown in Figure 8, which uses the same labeling introduced for Fig. 7. The cost ratio is normalized with respect to the minimum spanning tree (instead of the KMB tree). The error bars again represent 95% confidence interval. The cost ratio of BSMA with the delay bound set to that of the minimum-weight spanning tree is nearly identical to one. Thus the cost of the BSMA solution is nearly identical to that of the minimum-weight spanning tree solution, which is the optimal solution. By tightening the delay bound, a range of solutions can be obtained between the optimal cost solution and the minimum-delay solution. Note that, compared to KMB, the relative quality of the results obtained with BSMA improves with the number of destinations in the multicast group (this is apparent by comparing Figure 7 and Figure 8). Interestingly, it is not clear that the maximum k necessarily increases with the multicast membership for a given network size.

The results of BSMA using the greedy superedge selection were nearly identical (within 1%) to the naive selection. The results for four and $|V| - 1$ destinations are shown in Figure 11.

Figure 12 shows the cost ratio of BSMA in a graph of size 100, when the group sizes are varied over a large range. The results indicate similar behavior of BSMA for different group sizes.

BSMA was also applied using the Arpanet topology shown in Figure 13. The link cost and link delays were all set to equal 1. This choice of link cost and link delay tries to simulate the situation in which link costs and link delays are correlated. The number of group instances for each group size was at least 2500. Figure 14 shows the results using the Arpanet topology. The results agree with those obtained for the random graphs.

The delay bound given has a strong influence on the the maximum value of k in k-shortest-path computed by BSMA and thus on the execution time of BSMA. The value of k is a measure of the number of alternate paths considered to lower the cost of a superedge. A tighter delay bound limits the number of alternate lower-cost paths that can satisfy the delay bound. A tighter delay bound results in a larger k, because it is quite likely that most of the lower cost paths cannot satisfy the delay bound. For delay bounds larger than the minimum delay, the simulations show the maximum k to be small on average. Figure 9 shows

the average maximum value of k for different delay bounds. The results for the maximum k in the case when the whole network is one group is shown in Figure 10. It can be seen that a slight relaxation of the delay bound often results in considerably fewer computations. This is especially true for larger networks. The observed values of k needed when the delay bounds are loose is quite small. Nevertheless, the value of k can become a tuning knob, by specifying the maximum k as an input BSMA, to trade-off execution time of BSMA against minimal cost.

Another set of simulations was run to illustrate how the cost ratio can be traded against running time. In these simulations, the delay bound is set to the minimum possible for a given multicast group, because a tighter delay bound leads to larger values needed for k in order to minimize the tree cost (as can be observed in Figures 9 and 10). The simulations involved 12,000 instances of 30 member multicast groups in a 500-node graph. As Fig. 15 illustrates, setting larger limits on k decreases the cost ratio but increases the running time, and vice versa.

This relationship between the complexity of path-switchings used to minimize the cost of the tree (i.e., the value of k) and the amount of reductions in the cost obtained is quite interesting. Since larger values of k are needed when the delay bounds are tighter, the amount of cost reduction obtained with large values of k is not very significant compared to the cost reduction attained with small values. Larger values of k indicate larger cost alternate paths, thus the cost reduction from replacing a superedge with a new path becomes less. In other words, there is a diminishing benefit from using larger values of k in both cost reduction and running time. In practice, this means that the maximum value of k allowed in a run of BSMA can be fixed to a small value. In our experiments, values of k smaller than 5 were sufficient to render good cost reductions and small running times (Fig. 15).

The execution times of BSMA with the naive superedge selection and KMB on the random graphs used in the simulations are shown in Table 2. The times for the algorithms use the same labeling convention introduced for Fig. 7. The results are very encouraging, considering that the times are for execution on Sun Sparcstation 1+ and that the code is not optimized for speed at all. Notice that the execution times of `kmb` and `bsm-1` are nearly identical. This is ideal, because BSMA solves the relaxed problem as fast as KMB. As the problem becomes more constrained, BSMA naturally requires more time to solve it.

The running times of BSMA are much lower than the ones reported by Salama et al. [21]. Our simulation results for 30 members in a 100-node graph for the most stringent delay-bound are on the average about 4 times longer than KMB. This should be the worst-case scenario for BSMA in that the delay bound is tight and the group size is medium with respect to the network size. BSMA took on average 4.88 seconds and KMB took on average 1.11 sec on a 167 MHz UltraSPARC. The average number of k is 23. The execution time of BSMA is nevertheless on the same order of magnitude as KMB. We conjecture that the differences between our results on BSMA execution times and those reported by Salama et al. [21] are due to differences in the specific k-shortest path algorithm used to implement BSMA and the implementation of Dijkstra's algorithm. Our implementation of the k-shortest path also uses Dijkstra's algorithm implemented using a binary heap data structure. Our simulations were carried out on sparse graphs, that is, the number of edges m is $O(n)$. Using a binary heap for the priority queue gives the complexity of $O(n \log(n))$ for Dijkstra's algorithm in sparse graphs [7]. It is also worthwhile noting that the k-shortest path algorithm used in our own implementation of BSMA has time complexity of $O(kn^2 \log(n))$, while k-shortest path algorithms exist of complexity $O(kn \log(n))$; hence, BSMA could run even faster than our own results indicate.

Moreover, as demonstrated earlier, the execution time of BSMA can be traded off against the cost reductions by controlling the maximum value of k. Since there is a diminishing benefit from using larger values of k in both cost reduction and running time, the maximum value of k allowed in a run of BSMA can be set in advance to a small value.

6 Concluding Remarks

Multicast tree construction is becoming an integral part of multimedia application support. This paper proposed BSMA, an algorithm based on complete topology information for the construction of delay-bounded minimum-cost multicast trees. The contribution of our work lies both in the formulation of the problem and the novelty of the algorithm used to solve it. We allow variable delay bounds set for different destinations or different media; this simulates timing requirements of realistic networks supporting multimedia applications. BSMA minimizes the total link cost of the tree, while satisfying the delay constraints. Instead of using the one-pass growing of the multicast tree used in most previous works, BSMA uses

an iterative optimization process to further minimize the tree cost. The cost minimization is monotonically achieved after a series of delay bounded path-switching improvements. The simulation results show that BSMA can produce delay-bounded multicast trees that have low cost. Recent results by Salama et al. [21] confirm our results and show that BSMA is the best algorithm in terms of cost and number of sessions established for computation of delay-constrained minimum-cost multicast trees among all the constrained Steiner tree (CST) algorithms reported to date.

Additional work is needed to make BSMA or other CST heuristics scale to large-scale networks. We anticipate that CST heuristics will be applied to large-scale networks within the context of hierarchical routing, because the underlying routing mechanisms used to disseminate topology data require the aggregation of information in order to cope with network size (e.g., OSPF [17]). In addition to reducing the number of nodes and links that a node running BSMA needs to know, BSMA's running time can be made faster by limiting the number of k shortest paths used in BSMA's iterations, at the expense of larger tree costs.

The iterative approach introduced in BSMA can also be applied to related multicast problems, such as obtaining trees of smallest cost with end-to-end delay bounds and delay-variation bounds [20].

References

[1] B. Awerbuch, A. Baratz, and D. Peleg. Cost-sensetive analysis of communication protocols. In *Proc. ACM Symp. on Principles of Distributed Computing*, pages 177–187, 1990.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1992.

[3] K. Bharath-Kumar and J. Jaffe. Routing to Multiple Destination in Computer Networks. *IEEE Trans. on Comm.*, COM-31:343–351, 1983.

[4] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. A Survey of Optimization Techniques for Integrated Circuits. *Proc. of the IEEE*, 69(10):1334–1362, 1991.

[5] A. Cayley. A Theorem on Trees. *Quart. J. Math.*, 23:376–378, 1889.

[6] J. Cong, A.B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Provably Good Performance-Driven Global Routing. *IEEE Trans. on Computer-Aided Design*, CAD-11(6):739–752, 1992.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[8] E. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[9] M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. of ACM*, 34(3):596–615, 1987.

[10] J. J. Garcia-Luna-Aceves. Reliable Broadcast of Routing Information Using Diffusing Computations. In *Proc. of IEEE Globecom '92*, Dec. 1992.

[11] J. J. Garcia-Luna-Aceves and J. Behrens. Distributed, Scalable Routing Based on Vectors of Link States. *IEEE J. on Selected Areas in Comm.*, 13(8):1383–95, Oct. 1995.

[12] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Co., 1979.

[13] N. Katoh, T. Ibaraki, and H. Mine. An Efficient Algorithm for K Shortest Simple Paths. *Networks*, 12:411–427, 1982.

[14] V. P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicast Routing for Multimedia Communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[15] L. Kou, G. Markowsky, and L. Berman. A Fast Algorithm for Steiner Trees. *Acta Informatica*, 15:141–145, 1981.

[16] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[17] J. Moy. OSPF Version 2. *Internet Draft, RFC-1247*, Jul. 1991.

[18] R. Prim. Shortest Connection Networks and Some Generalizations. *Bell Systems Tech. J.*, 36:1389–1401, 1957.

[19] V. Rayward-Smith. The Computation of Nearly Minimal Steiner Trees in Graphs. *Intl. J. Math. Educ. Sci. Tech.*, 14(1):15–23, 1983.

[20] G. N. Rouskas and I. Baldine. Multicast Routing with End-to-End Delay and Delay Variation Constraints. *IEEE J. on Selected Areas in Comm.*, 15(3):346–356, April 1997.

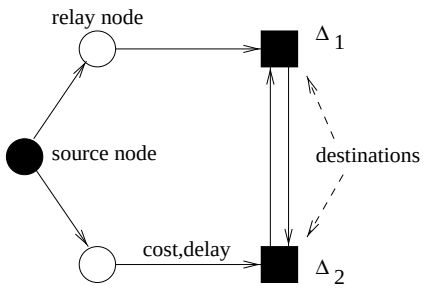
[21] H. F. Salama, D. S. Reeves, and Y. Viniotis. Evaluation of Multicast Routing Algorithms for Real-Time Communications on High-Speed Networks. *IEEE J. on Selected Areas in Comm.*, 15(3):332–345, Apr. 1997.

[22] H. Takahashi and A. Matsuyama. An Approximate Solution for the Steiner Problem in Graphs. *Math. Japonica*, 6:573–577, 1990.

[23] B. Waxman. Routing of Multipoint Connections. *IEEE J. on Selected Areas in Comm.*, 6:1617–1622, December 1988.

[24] Jin Y. Yen. Finding the k Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, 1971.

[25] Q. Zhu and W.M. Dai. Delay-Bounded Steiner Tree Algorithm for Performance-Driven Layout. *Technical Report, UCSC-CRL-93-46, University of California, Santa Cruz*, Oct. 1993.



link cost = cost metric, such as bandwidth utilization or dollar cost

link delay = queuing delay + transmission delay + propagation delay

Δ_i = delay bound from the source to destination i

Figure 1: Network model

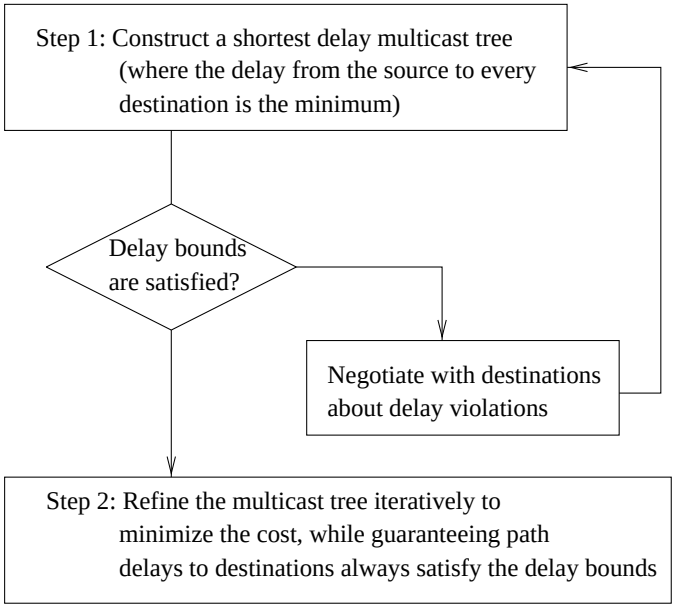
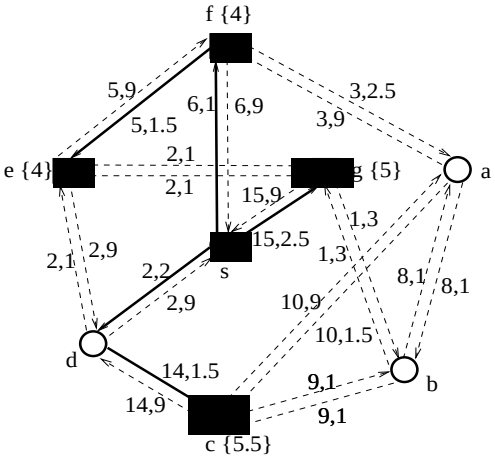
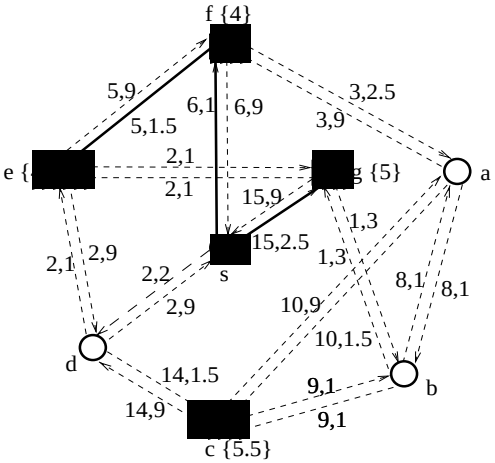


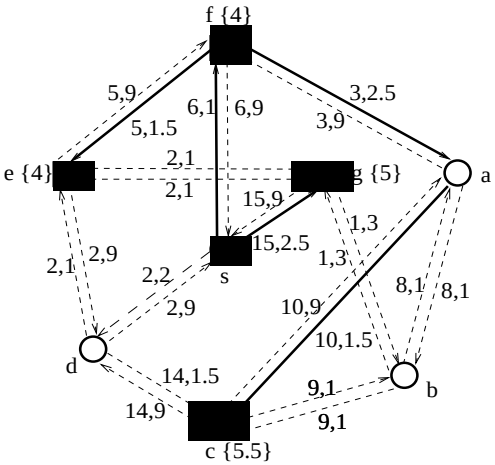
Figure 2: Methodology used by BSMA.



(a) Shortest delay tree (T_0)



(b) Removing superedge $\overline{sc}$



(c) Adding superedge $\overline{fc}$

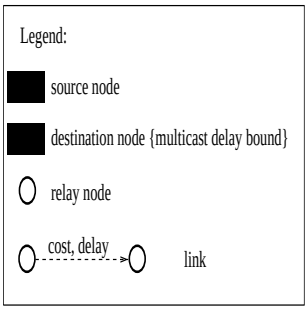


Figure 3: Delay-bounded path-switching improvement. (a) Removing the highest-cost superedge $\langle s, d, c \rangle$ from T_0; (b) T_0 is partitioned to two disjoint trees T_0^1 and T_0^2; (c) Reconnecting T_0^1 and T_0^2 by the delay-bounded shortest path $\langle f, a, c \rangle$ between the two trees gives T_1.

```
INPUT:
  G(V, E) = graph, s = source node,
  D = set of destination nodes,
  DB = set of delay bounds for destination nodes,
  T_j = the tree at iteration j.
OUTPUT:
  A delay bounded Steiner tree spanning D ∪ {s}.
PROCEDURE MulticastTree(G(V, E), s, D, DB) {
  j ← 0;
  T_j ← minimum-delay tree spanning D ∪ {s} by
    Dijkstra's algorithm;
  Initialize all superedges of T_j as unmarked;
  loop {
    p_h ← the highest-cost unmarked superedge
      among all unmarked superedges in T_j;
    if (p_h = NULL) then
      Return;
    Mark superedge p_h;
    Remove p_h from T_j, to obtain T_j^1 and T_j^2;
    p_s ← delay-bounded shortest path between T_j^1 and T_j^2.
    T_{j+1} ← p_s ∪ T_j^1 ∪ T_j^2;
    if (p_s ≠ p_h)
      Unmark all marked superedges;
    j ← j + 1;
  }
}
```

Figure 4: High-Level Description of BSMA.

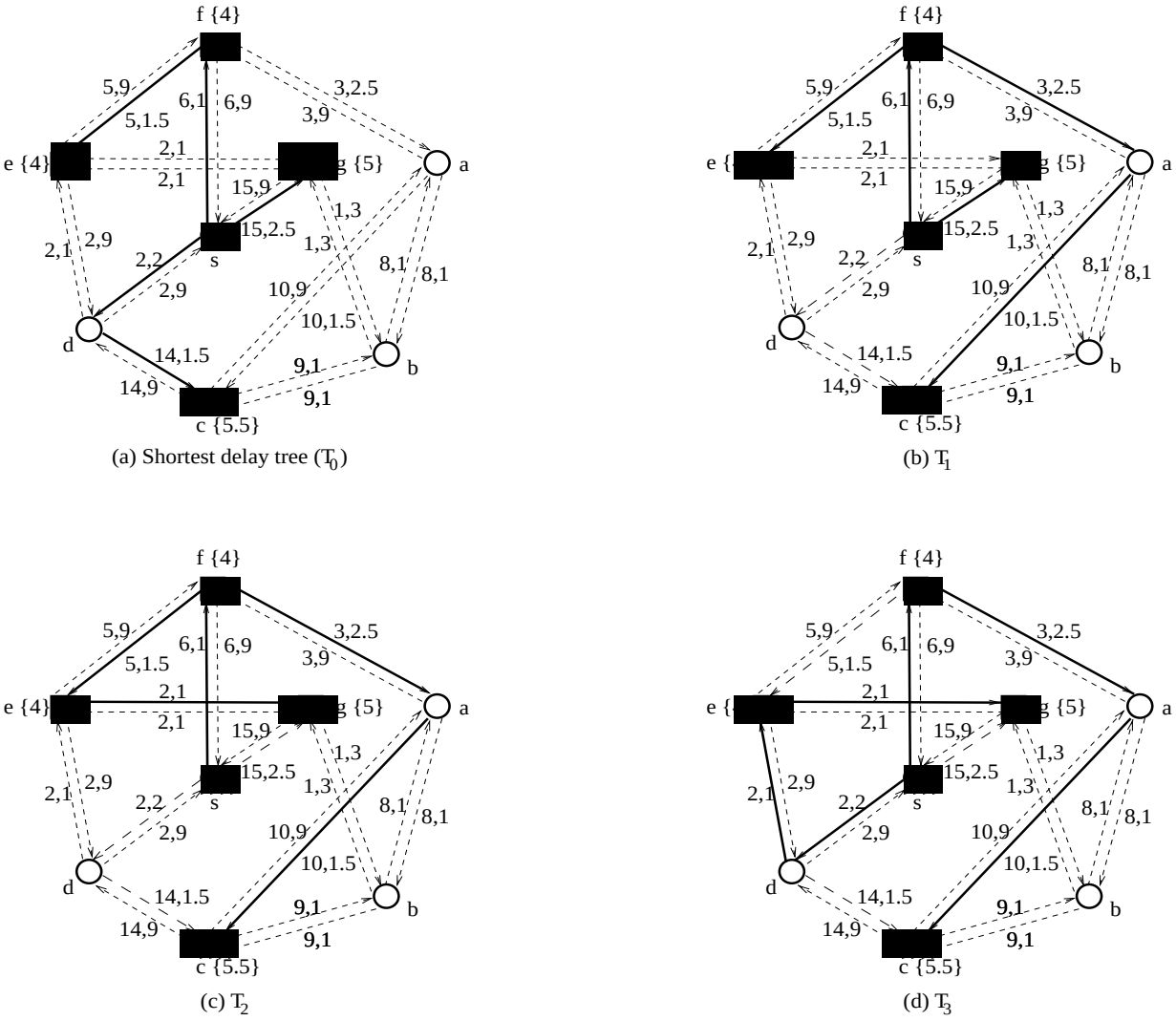


Figure 5: Utilization-Driven Multicast Tree Construction and Minimization

```
INPUT:
  G(V,E) = graph, s = source node,
  D = set of destination nodes,
  DB = set of delay bounds for destination nodes,
  T_j = the tree at iteration j,
OUTPUT:
  A delay bounded Steiner tree spanning D ∪ {s}.
PROCEDURE GreedyMulticastTree(G(V,E), s, D, DB) {
  j ← 0;
  T_j ← minimum-delay tree spanning D ∪ {s}
    by Dijkstra's algorithm;
  do {
    g* ← 0;
    foreach (superedge p of T_j) {
      Remove p from T_j to get T_j^1 and T_j^2;
      q ← delay bounded shortest path between T_j^1 and T_j^2;
      c_p ← cost of p;
      c_q ← cost of q;
      g ← c_p − c_q;
      if (g* < g) then {
        g* ← g;
        p_best ← p;
        q_best ← q;
      }
      T_j ← p ∪ T_j^1 ∪ T_j^2;
    }
    if (g* > 0) then {
      Remove p_best from T_j to get T_j^1 and T_j^2;
      T_{j+1} ← q_best ∪ T_j^1 ∪ T_j^2;
      j ← j + 1;
    }
  } while (g* > 0);
}
```

Figure 6: A Greedy Variant of BSMA.

$\|V\|$	average degree
25	3.76
50	5.04
75	4.88
100	5.00

Table 1: Average degree for the networks in the simulations.

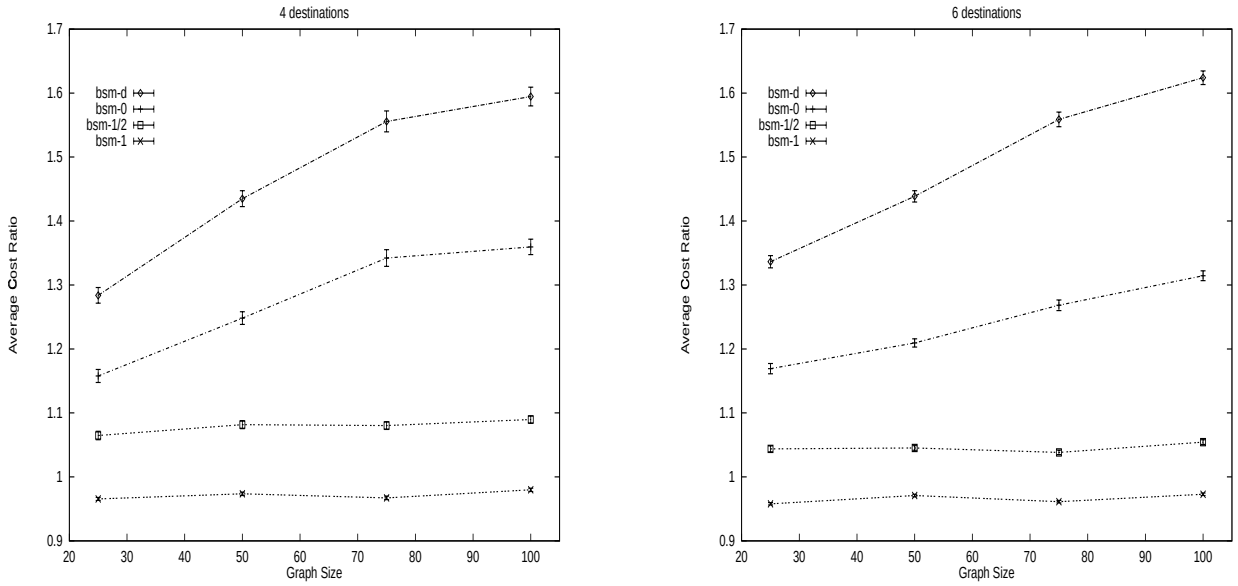


Figure 7: Network cost of multicast tree as a function of graph size using the naive superedge selection.

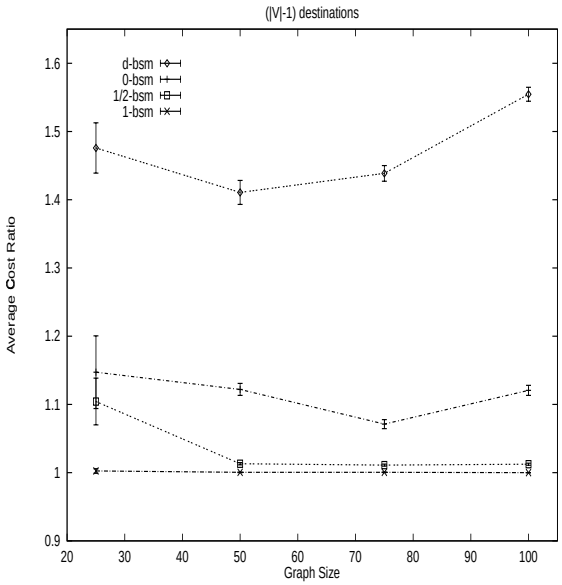


Figure 8: Average network cost when all nodes are in a group.

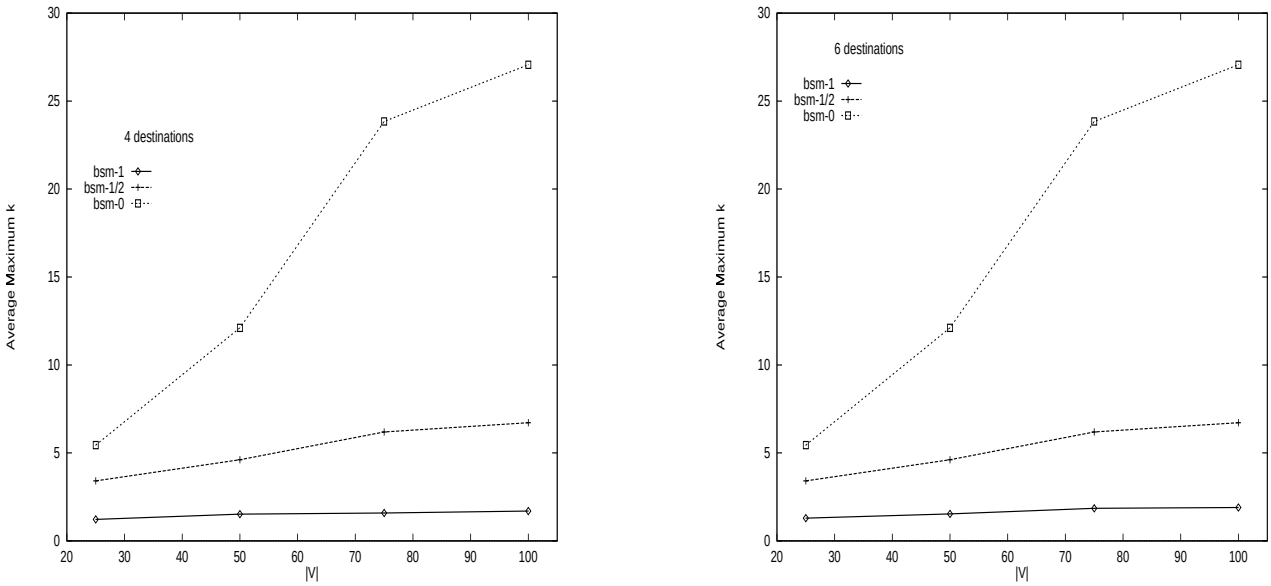


Figure 9: The average maximum number of k-shortest paths computed to obtain constrained tree. As the delay bound is tightened, larger values of k are computed.

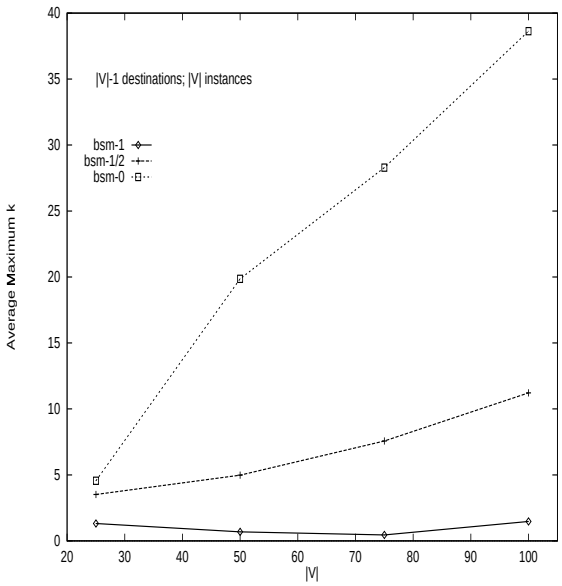


Figure 10: The average maximum number of k-shortest paths computed to obtain constrained tree. As the delay bound is tightened, larger values of k are computed.

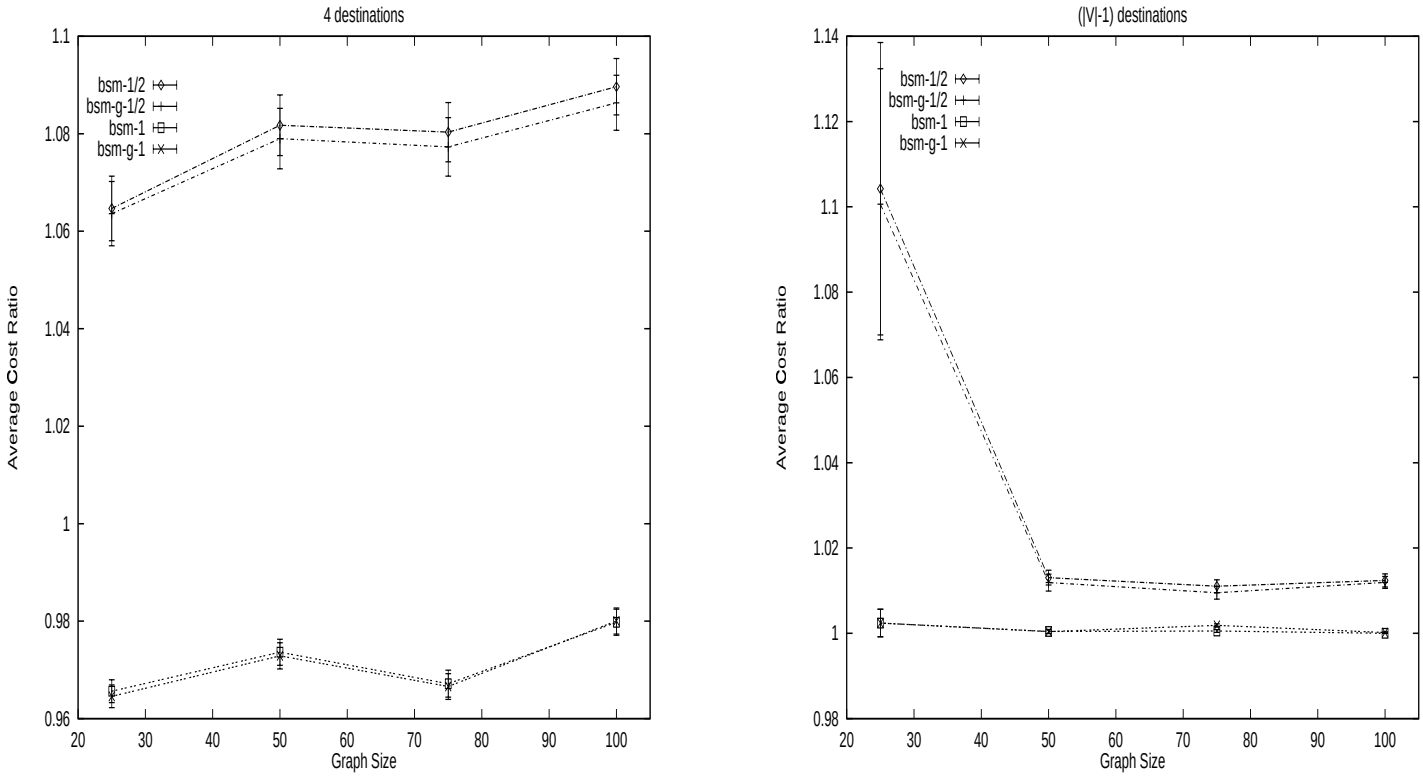


Figure 11: The comparison of minimization by greedy and naive superedge selection

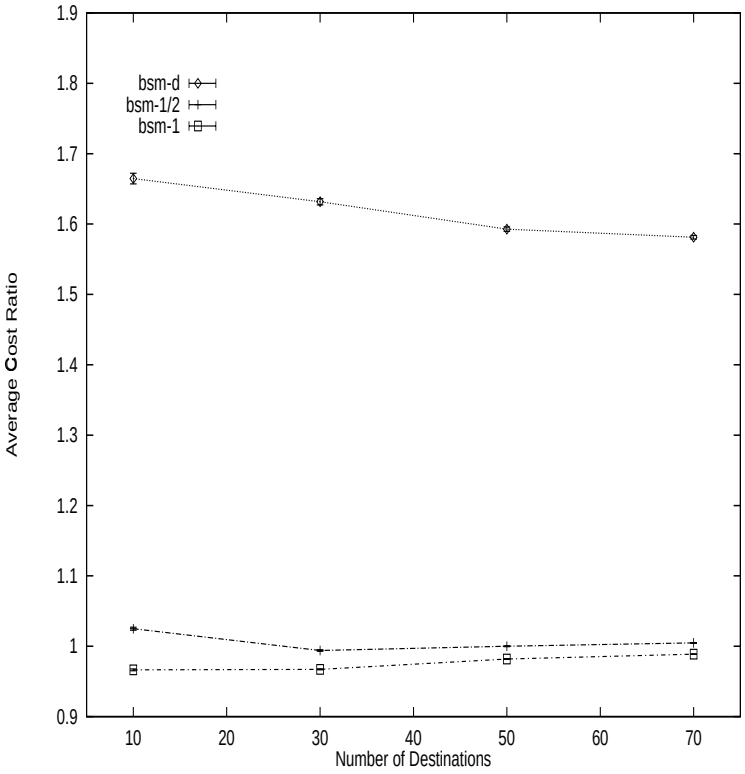


Figure 12: Different group sizes in 100-node graphs.

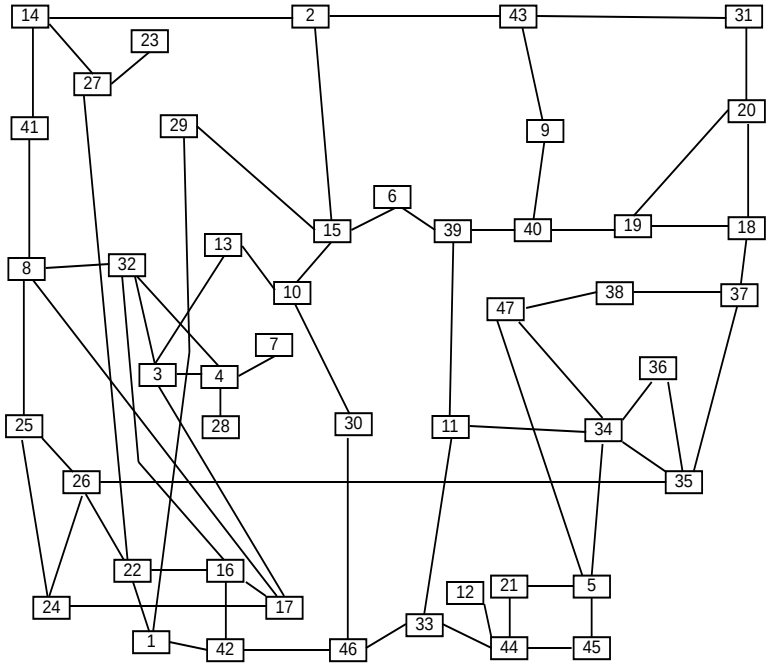


Figure 13: Arpanet topology.

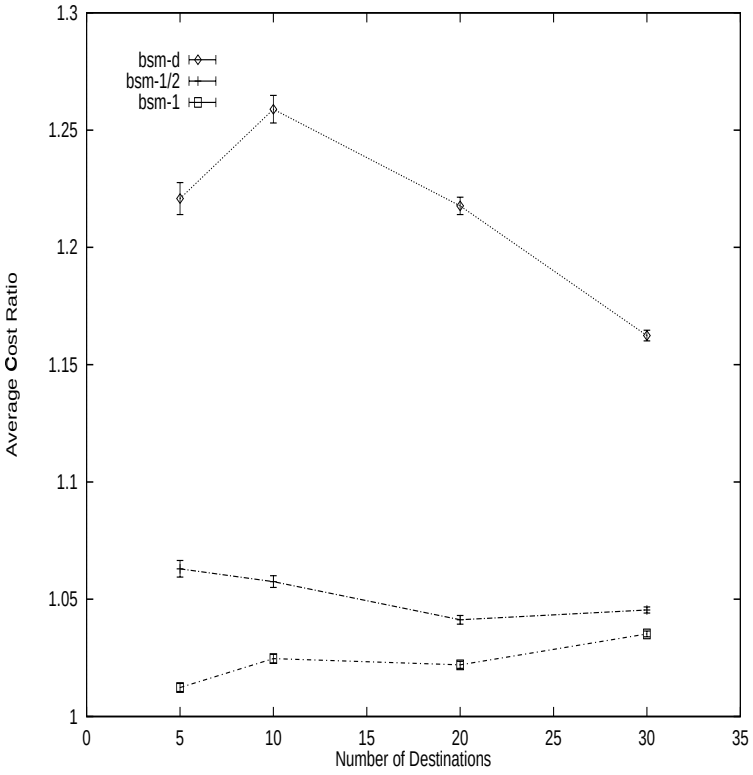


Figure 14: Different group sizes in the Arpanet.

$\mid V \mid$	$\mid D \mid = 4$			$\mid D \mid = 6$		
	kmb	bsm-1	bsm-1/2	kmb	bsm-1	bsm-1/2
25	0.2	0.2	0.3	0.3	0.3	0.4
50	0.5	0.5	0.8	0.7	0.7	1.1
75	0.7	0.8	1.7	1.1	1.2	2.1
100	0.9	1.1	2.8	1.4	1.7	3.4

Table 2: Average execution time per group, given in seconds, for naive BSMA and KMB on Sun Sparc1+ workstations.

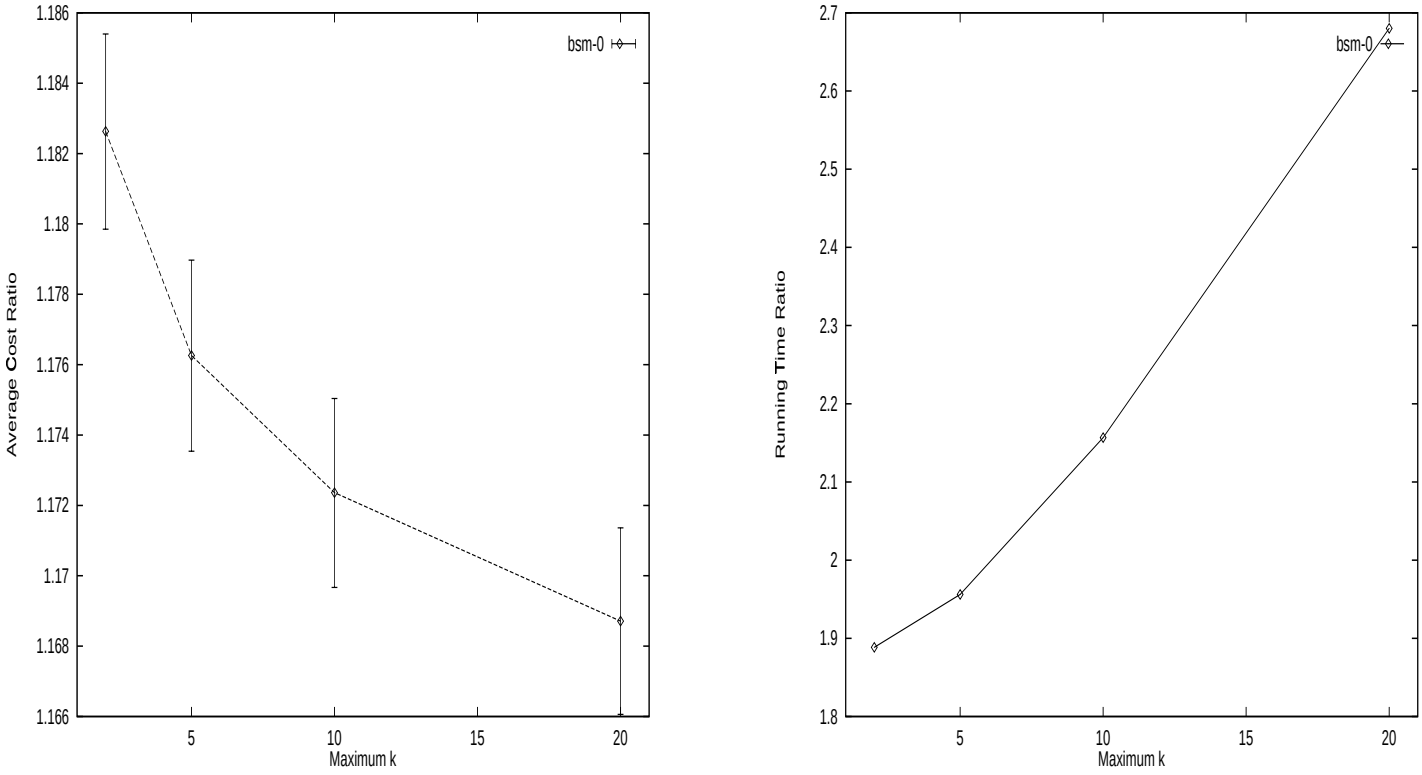


Figure 15: Trade-off of cost versus running time for a group of size 30 in a 500-node graph.